Base goals and knowledge:

Current Research Goals:

- Varying k8s Deployment Strategies
  - https://blog.container-solutions.com/kubernetes-deployment-strategies
    Single Service Ingress, Simple fanout, Name based virtual hosting
- Helm Packages
- Automating Helm upon creation.
  - https://www.baeldung.com/kubernetes-helm
  - https://github.com/RehanSaeed/Helm-Cheat-Sheet
  - https://opensource.com/article/20/1/automating-helm-deployments-bash
- Template strategies, and minimal requirements.

Required Features for a minimum pipelined k8s cluster:
-deployments
- monlith/micro sa port setup. Exposed ports, naming
-services
- default/custom service optional, port setup, naming
-ingress
- Optional,
-messaging server
-Redis only? Ports available, how many subscribers/which pods, communication pipeline,
-Helm dependencies
- optional, number of charts, location they should be placed, which pods have dependencies for them. tagging

Important to understand difference between helm and ansible:
Helm is a package manager for K8s. Ansible is a configuration management solution for mutable infrastructure (e.g. making sure that all your security updates are always installed on your machines)

Helm Commands:

Common Uses:

- helm search: Searches for charts
- helm pull: download a chart to local directory to view
- helm install: upload the chart to Kubernetes
- helm list: list releases of charts

Configuring environment variables with helm
By default, the default directories depend on the Operating System. The defaults are listed below:

- Linux:
    - Cache Path
        - &HOME/ .cache/helm
    - Configuration Path
        - &HOME/ .config/helm
    - Data Path
        - &HOME/ .local/share/helm
- Windows
    - Cache Path
        - %TEMP%\helm
    - Configuration Path
        - %APPDATA%\helm
    - Data Path
        - %APPDATA%\helm

Helm stores configuration based on the XDG base directory specification, so

- cached files are stored in $XDG_CACHE_HOME/helm
- configuration is stored in $XDG_CONFIG_HOME/helm
- data is stored in $XDG_DATA_HOME/helm

Name	Description
$XDG_CACHE_HOME	set an alternative location for storing cached files.
$XDG_CONFIG_HOME	set an alternative location for storing Helm configuration.
$XDG_DATA_HOME	set an alternative location for storing Helm data.
$HELM_DRIVER	set the backend storage driver. Values are: configmap, secret, memory, postgres
$HELM_DRIVER_SQL_CONNECTION_STRING	set the connection string the SQL storage driver should use.
$HELM_NO_PLUGINS	disable plugins. Set HELM_NO_PLUGINS=1 to disable plugins.
$KUBECONFIG	set an alternative Kubernetes configuration file (default "~/.kube/config")

More in-depth information can be found in the Helm Docs

- helm completion - generate autocompletions script for the specified shell (bash or zsh)
- helm create - create a new chart with the given name
- helm dependency - manage a chart's dependencies
- helm env - helm client environment information
- helm get - download extended information of a named release
- helm history - fetch release history
- helm install - install a chart
- helm lint - examine a chart for possible issues
- helm list - list releases
- helm package - package a chart directory into a chart archive
- helm plugin - install, list, or uninstall Helm plugins
- helm pull - download a chart from a repository and (optionally) unpack it in local directory
- helm repo - add, list, remove, update, and index chart repositories
- helm rollback - roll back a release to a previous revision
- helm search - search for a keyword in charts
- helm show - show information of a chart

- helm status - display the status of the named release
- helm template - locally render templates
- helm test - run tests for a release
- helm uninstall - uninstall a release
- helm upgrade - upgrade a release
- helm verify - verify that a chart at the given path has been signed and is valid
- helm version - print the client version information

Helm File Structure:

Helm create <wordpress> (To create initial file structure)

```
wordpress/
  Chart.yaml          # A YAML file containing information about the chart

  LICENSE             # OPTIONAL: A plain text file containing the license for the chart

  README.md           # OPTIONAL: A human-readable README file

  values.yaml         # The default configuration values for this chart

 values.schema.json  # OPTIONAL: A JSON Schema for imposing a structure on the
                              values.yaml file
  charts/             # A directory containing any charts upon which this chart depends.
  crds/               # Custom Resource Definitions
  templates/          # A directory of templates that, when combined with values,
                              will generate valid Kubernetes manifest files.
  templates/NOTES.txt # OPTIONAL: A plain text file containing short usage notes
```

Chart.Yaml: see notes below

2 types: application and library

Values.yaml:

Values for the templates are supplied two ways:

- Chart developers may supply a file called values.yaml inside of a chart. This file can contain default values.
- Chart users may supply a YAML file that contains values. This can be provided on the command line with helm install.

When a user supplies custom values, these values will override the values in the chart's values.yaml file

Predefined Values

Values that are supplied via a values.yaml file (or via the --set flag) are accessible from the .Values object in a template. But there are other pre-defined pieces of data you can access in your templates.

The following values are pre-defined, are available to every template, and cannot be overridden. As with all values, the names are *case sensitive*.

- Release.Name: The name of the release (not the chart)
- Release.Namespace: The namespace the chart was released to.
- Release.Service: The service that conducted the release.
- Release.IsUpgrade: This is set to true if the current operation is an upgrade or rollback.
- Release.IsInstall: This is set to true if the current operation is an install.
- Chart: The contents of the Chart.yaml. Thus, the chart version is obtainable as Chart.Version and the maintainers are in Chart.Maintainers.
- Files: A map-like object containing all non-special files in the chart. This will not give you access to templates, but will give you access to additional files that are present (unless they are excluded using .helmignore). Files can be accessed using {{ index .Files "file.name" }} or using the {{ .Files.Get name }} function. You can also access the contents of the file as []byte using {{ .Files.GetBytes }}
- Capabilities: A map-like object that contains information about the versions of Kubernetes ({{ .Capabilities.KubeVersion }} and the supported Kubernetes API versions ({{ .Capabilities.APIVersions.Has "batch/v1" }})

NOTE: Any unknown Chart.yaml fields will be dropped. They will not be accessible inside of the Chart object. Thus, Chart.yaml cannot be used to pass arbitrarily structured data into the template. The values file can be used for that, though.

Values files

Considering the template in the previous section, a values.yaml file that supplies the necessary values would look like this:

```
imageRegistry: "quay.io/deis"
dockerTag: "latest"
pullPolicy: "Always"
storage: "s3"
```

Library charts
https://helm.sh/docs/topics/library_charts/
These can be used as dependencies charts for other Helm setups.

Using ECR:

Amazon ECR requires that users have allow permissions to the ecr:GetAuthorizationToken API through an IAM policy before they can authenticate to a registry and push or pull any images from any Amazon ECR repository.
Tag your image with the Amazon ECR registry, repository, and optional image tag name combination to use. The registry format is aws_account_id.dkr.ecr.region.amazonaws.com. The repository name should

match the repository that you created for your image. If you omit the image tag, we assume that the tag is latest.

The following example tags an image with the ID e9ae3c220b23 as aws_account_id.dkr.ecr.region.amazonaws.com/my-web-app

Standard
-docker images (get a list of all images)

-Docker build -t tag ./path

-docker tag (ie e9ae3c220b23)
aws_account_id.dkr.ecr.region.amazonaws.com/my-web-app-
-docker push aws_account_id.dkr.ecr.region.amazonaws.com/my-web-app

Helm Charts .yaml:

https://helm.sh/docs/chart_best_practices/conventions/
https://helm.sh/docs/topics/charts/

A look at all fields of a chart.yaml:

- **apiVersion**: The chart API version (required)
- **name**: The name of the chart (required)
    - Chart names should be lower case letters and numbers. Words may be separated with dashes (-) Examples:
        - drupal
        - nginx-lego
        - aws-cluster-autoscaler

- The directory that contains a chart **MUST** have the same name as the chart. Thus, the chart nginx-lego MUST be created in a directory called nginx-lego/. This is not merely a stylistic detail, but a requirement of the Helm Chart format.
- **version**: A SemVer 2 version (required)https://semver.org/spec/v2.0.0.html

  The chart's version. This version number should be incremented each time you make changes to the chart and its templates, including the app version.
- **kubeVersion**: A SemVer range of compatible Kubernetes versions (optional)
- **description**: A single-sentence description of this project (optional)
- **type**: It is the type of chart (optional) A chart can either be an application or a library chart.
    - Application charts are a collection of templates that can be packaged into versioned archives to be deployed.
    - Library charts provide useful utilities or functions for the chart developer. They're included as a dependency of application charts to inject those utilities and functions into the rendering pipeline. Library charts do not define any templates and therefore cannot be deployed.
- **keywords**:
- - A list of keywords about this project (optional)
- **home**: The URL of this projects home page (optional)
- **sources**:
- - A list of URLs to source code for this project (optional)
- **dependencies**: *# A list of the chart requirements (optional)*
- - **name**: The name of the chart (nginx)

- **version**: The version of the chart ("1.2.3")
- **repository**: The repository URL ("https://example.com/charts") or alias ("@repo-name")
- **condition**: (optional) A yaml path that resolves to a boolean, used for enabling/disabling charts (e.g. subchart1.enabled)
- **tags**: # *(optional)*
- - Tags can be used to group charts for enabling/disabling together
- **enabled**: (optional) Enabled bool determines if chart should be loaded
- **import-values**: # *(optional)*
- - ImportValues holds the mapping of source values to parent key to be imported. Each item can be a string or pair of child/parent sublist items.
- **alias**: (optional) Alias usable alias to be used for the chart. Useful when you have to add the same chart multiple times
- **maintainers**: # *(optional)*
- - **name**: The maintainers name (required for each maintainer)
- **email**: The maintainers email (optional for each maintainer)
- **url**: A URL for the maintainer (optional for each maintainer)
- **icon**: A URL to an SVG or PNG image to be used as an icon (optional).
- **appVersion**: The version of the app that this contains (optional). This needn't be SemVer. Seperate from the chart version. This version number should be incremented each time you make changes to the application. The app and chart

versions can be incremented together or separately spending on which convention you want to use just make sure you stick to it.

- **deprecated**: Whether this chart is deprecated (optional, boolean)
- **annotations**:
- **example**: A list of annotations keyed by name (optional).

Playbook Integration:

Important to understand difference between helm and ansible: Helm is a package manager for K8s. Ansible is a configuration management solution for mutable infrastructure (e.g. making sure that all your security updates are always installed on your machines)

Presentation Preparation:

1: Tasks and Goals.

Ensure that K8s could be run with helm, and be deployed by Spinnaker using helm's charts, and be both functional and streamlined in as much of a way as possible.

2: Biggest Challenges

Caliber itself was an exceptionally difficult program to work around when many of the caveats of the program were not well documented. Further, the program's tests themselves Were often unrunnable.

Another large challenge we faced was the design of Spring applications in the presence of Kubernetes, when Most of us had never worked with the technology at all.

Learning {{GO}} to add dynamic functionalities to the setups in yaml files.

Navigating The Documentation for helm and K8s to set up deployment.

3: Hurdles Overcome

- Implementation of {{GO}}.
- Building Templates and Charts for the cluster's successful deployment.
- Manipulating the yaml files to match the configuration of caliber.

What could be done with more time?

1. Bash script that can be run on Ansible to configure custom YAML files for specific Micro-Service Architectures. IE
    a. Deployments- # of deployments, # of Replicas, Exposed ports, Image name and location.
    b. Service- Service type, ports exposed
    c. Ingress
    d. Configmaps
    e. Messaging

4: live Presentation